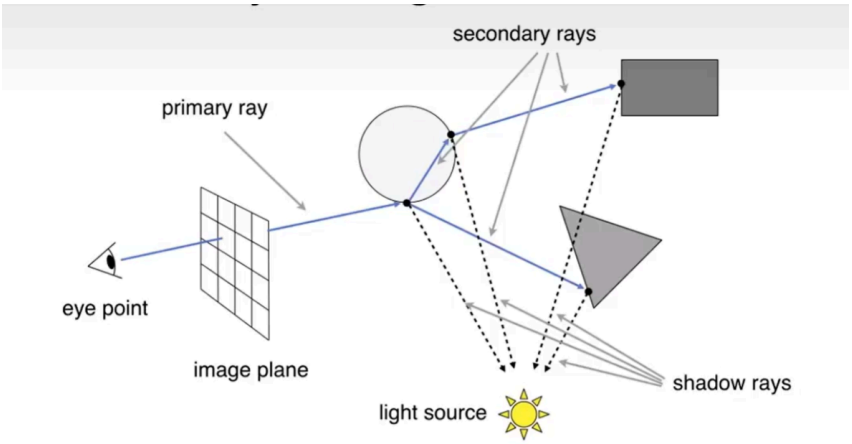
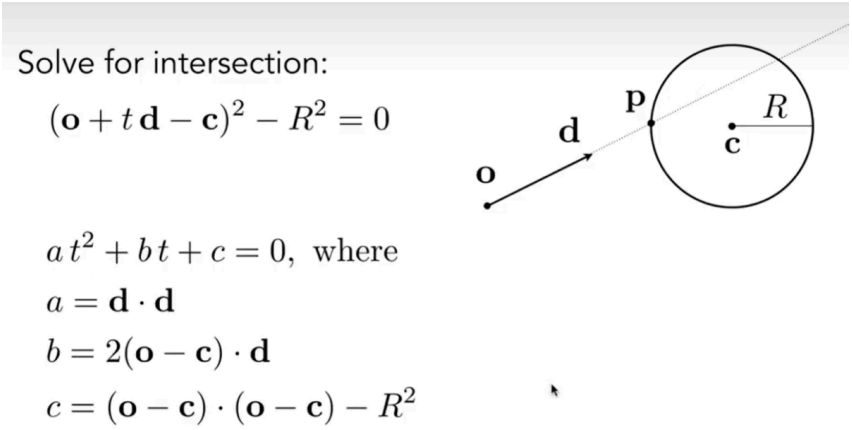
光线追踪!

光路可逆，嗯。电眼逼人。

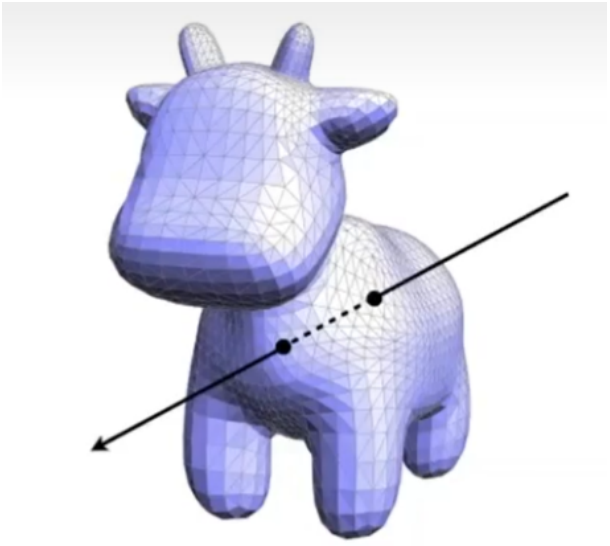


之前看过一些光线追踪的演示，可以设定光线允许的弹射次数，不同的弹射次数看起来效果很不一样。



这里定义的光线（射线）是一个关于时间的位置函数。

还得是牛牛。其实对于隐式的东西我暂时不太感兴趣。



课上把三维中光线与三角形相交的问题转化为光线与平面相交+平面上的点与平面上的三角形相交两个子问题，并用点法式定义平面。

还提到了一种 MT 算法，用到了之前提到的三角形重心坐标。

Gramer's Rule 看着就很 high-cost，感觉不如消元法。

Möller Trumbore Algorithm

A faster approach, giving barycentric coordinate directly

Derivation in the discussion section!

$$\vec{O}+t\vec{D}=(1-b_1-b_2)\vec{P}_0+b_1\vec{P}_1+b_2\vec{P}_2$$

Where:

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\vec{S}_1 \cdot \vec{E}_1} \begin{bmatrix} \vec{S}_2 \cdot \vec{E}_2 \\ \vec{S}_1 \cdot \vec{S} \\ \vec{S}_2 \cdot \vec{D} \end{bmatrix}$$

$$\vec{E}_1 = \vec{P}_1 - \vec{P}_0$$
$$\vec{E}_2 = \vec{P}_2 - \vec{P}_0$$
$$\vec{S} = \vec{O} - \vec{P}_0$$
$$\vec{S}_1 = \vec{D} \times \vec{E}_2$$
$$\vec{S}_2 = \vec{S} \times \vec{E}_1$$

Recall: How to determine if the "intersection" is inside the triangle?

Hint:
(1-b1-b2), b1, b2 are barycentric coordinates!

Cost = (1 div, 27 mul, 17 add)

加速！

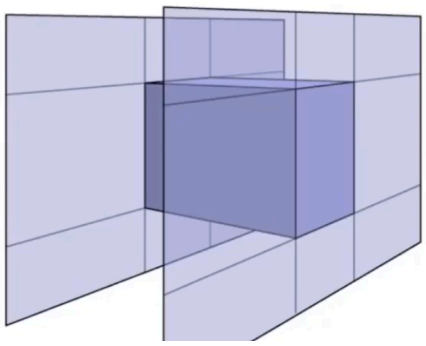
Bounding Volumes 包围体积（包围盒）

Understanding: **box is the intersection of 3 pairs of slabs**

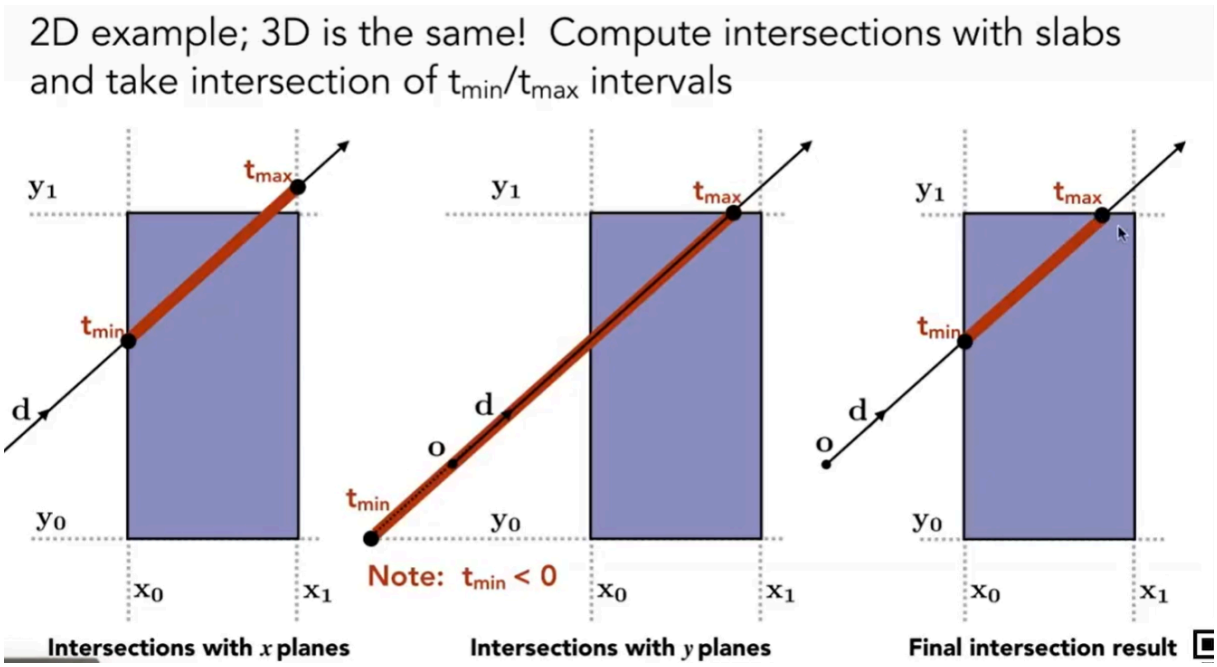
Specifically:

We often use an **Axis-Aligned Bounding Box (AABB)**
(轴对齐包围盒)

i.e. any side of the BB is along either x, y, or z axis



这种 AABB 包围盒有个好处是用六个数对就可以表示。且光线与这类平面求交更方便。



这个图如何理解呢，$t_{\min}$ 和 $t_{\max}$ 本质是定义了光线与一对对面的包围体积的交集。那么盒子的内部如何定义呢，其实就是所有对面的包围体积的交集。

在盒子 and 在光线内 = 在所有对面的包围体积的交集 and 在光线 =（在光线 and 在 x 对面的包围体积）and（在光线 and 在 y 对面的包围体积）and（在光线 and 在 z 对面的包围体积）

然后这个具体的求法就是下面的这个图了，说实话这个 key idea 我不太懂，但你理解成线段求交公式也是可以的。

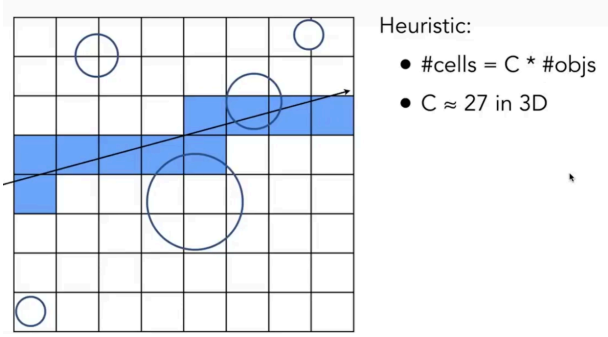
- Recall: a box (3D) = three pairs of infinitely large slabs
- Key ideas
  - The ray enters the box **only when** it enters all pairs of slabs
  - The ray exits the box **as long as** it exits any pair of slabs
- For each pair, calculate the t_{min} and t_{max} (negative is fine)
- For the 3D box, t_{enter} = **max**{t_{min}}, t_{exit} = **min**{t_{max}}

嗯，边界条件课上已经给了，我就不细说了。

- However, ray is not a line
  - Should check whether t is negative for physical correctness!
- What if $t_{exit} < 0$?
  - The box is “behind” the ray — no intersection!
- What if $t_{exit} >= 0$ and $t_{enter} < 0$?
  - The ray’s origin is inside the box — have intersection!
- In summary, ray and AABB intersect iff
  - $t_{enter} < t_{exit}$ && $t_{exit} >= 0$

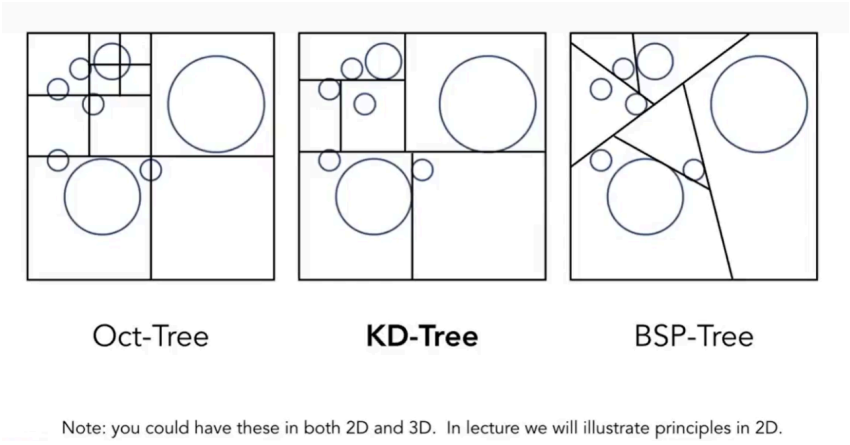
继续谈论加速。

另一种方法是对空间分块，每个块分成有物体还是没物体。



Spatial Partitions 空间划分

进一步的分块，每块不一定是均匀的。（牢 OIer 再看到 KD-Tree 稍微有些感慨）

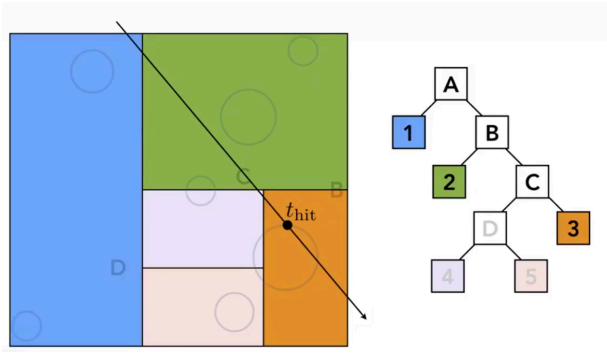


二叉树/四叉树/八叉树基本就是那样吧。

KD-Tree 是纯纯的二叉树，而且用的是 AABB 型包围盒。

BSP-Tree 不太好做与光线的相交计算。

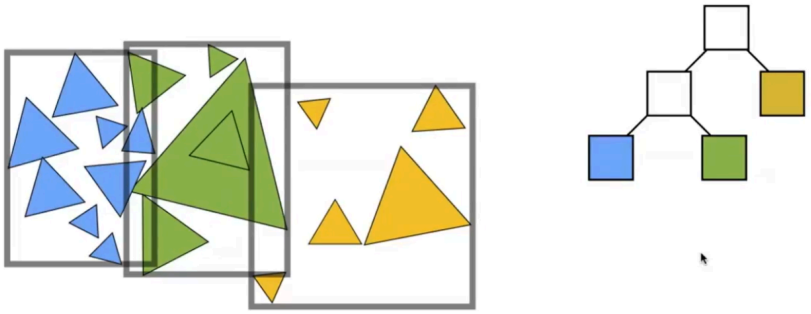
KD-Tree



KD-Tree 依然只做到了相对快速地找到了“可能与光线相交的物体”，但考虑到一般不太用隐式表示来描述物体的表面而是用三角形，所以后面应该还会提到更多优化。

嗯，判定三角形与平面的交集是个很难的问题。

Object Partitions and BVH (Bounding Volume Hierarchies)
如图，它也解决了如何快速找到与物体相交的三角形。



但代价在于包围盒会相交，BVH 相关的研究也有很多集中在优化划分方式以减少相交这方面。

提到了两个优化，第一个保证了每个叶子节点的 Bounding Volumes 尽量在空间中均匀，第二个则是保证了叶子节点内的三角形数量尽量均匀。

How to subdivide a node?

- Choose a dimension to split
- Heuristic #1: Always choose the longest axis in node
- Heuristic #2: Split node at location of **median** object

还提到了怎么快速得出中位三角形，也就是如何快速做 kth 问题，答案是快速选择算法。之后再还债吧（悲

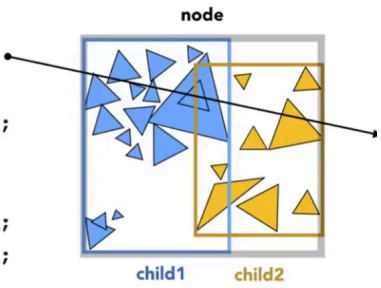
动态场景不好预处理，嗯，所以光线追踪的开销非常大。

```
Intersect(Ray ray, BVH node) {
  if (ray misses node.bbox) return;

  if (node is a leaf node)
     test intersection with all objs;
     return closest intersection;

  hit1 = Intersect(ray, node.child1);
  hit2 = Intersect(ray, node.child2);

  return the closer of hit1, hit2;
}
```



徐导曾经说过一句话，大意是一个导演的表达都在片子里，哈哈。

所以也可以说程序员的表达都在代码里，嗯。

这里顺便畅想一下，嗯，KD-Tree 和 BVH 展现的问题归根结底是我们要采用 AABB 形式的包围盒，而 DeepLearning 乃至类脑计算元件如果能对几何结构更复杂的 Bounding Volumes 也能有很好的效果的话……哈哈说笑了，虽然我们梦里大脑渲染出来的场景都很清晰，但可能采用了完全不同的光线追踪管线。

Basic Radiometry 辐射度量学

这个看名字就非常 Physical，是路径追踪的基础（前面提的某个风格光线追踪的 pro-max）。

老师说单词不太好翻译，那么拜托了我的 LLM 猫娘。

翻译总结表

英文术语	新译名	核心逻辑
radiant flux	辐能率	辐射能量的时间速率（功率）
intensity	辐角密	单位立体角内的辐射能量密度
irradiance	辐照密	单位面积接收的辐射能量密度
radiance	辐角照密	单位立体角与单位投影面积的综合能量密度

当然可能并不完全准确，还是结合课程我自己来修改一下吧。

Definition: Radiant energy is the energy of electromagnetic radiation. It is measured in units of joules, and denoted by the symbol:

$$Q \ [\mathrm{J} = \mathrm{Joule}]$$

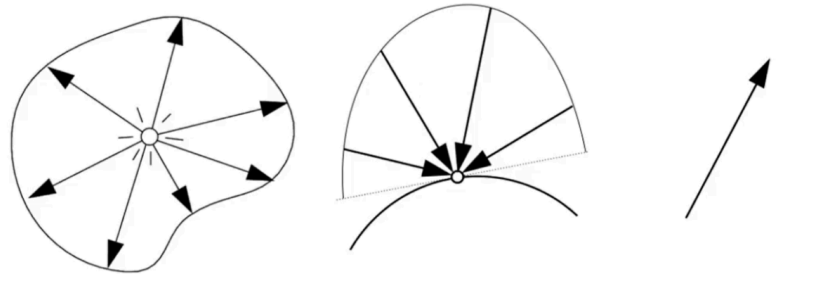
Definition: Radiant flux (power) is the energy emitted, reflected, transmitted or received, per unit time.

$$\Phi \equiv \frac{\mathrm{d}Q}{\mathrm{d}t} \ [\mathrm{W} = \mathrm{Watt}] \ [\mathrm{lm} = \mathrm{lumen}]^{*}$$

首先 Radiant flux 确实翻译成辐能率很合适，不过稍微考虑一下还是翻译成 **辐射能率**。

辐能率的单位除了 W 瓦特还有 lumen 流明，lumen 是一个光学中的单位。

接下来是其它三个词。



Light Emitted From A Source

"Radiant Intensity"

Light Falling On A Surface

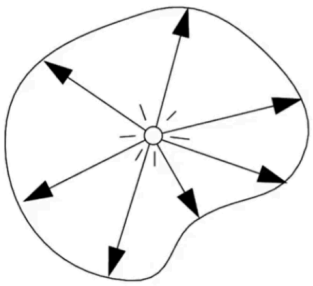
"Irradiance"

Light Traveling Along A Ray

"Radiance"

Definition: The radiant (luminous) intensity is the power per unit **solid angle (?)** emitted by a point light source.

(立体角)



$$I(\omega) \equiv \frac{\mathrm{d}\Phi}{\mathrm{d}\omega}$$

$$\left[\frac{\mathrm{W}}{\mathrm{sr}}\right] \left[\frac{\mathrm{lm}}{\mathrm{sr}} = \mathrm{cd} = \mathrm{candela}\right]$$

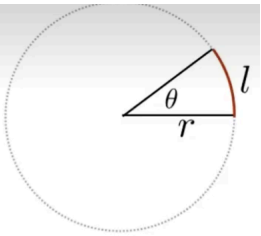
The candela is one of the seven SI base units.

intensity 翻译成 **角辐射能率** 更合适，毕竟辐角是个已有名词，会过度激活记忆导致头晕。

candela 还是个标准单位吧。

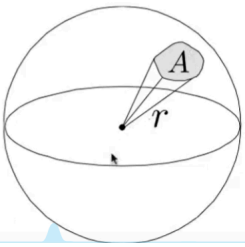
Angle: ratio of subtended arc length on circle to radius

- $\theta = \frac{l}{r}$
- Circle has 2π radians



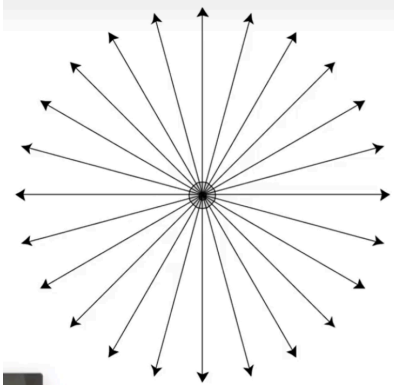
Solid angle: ratio of subtended area on sphere to radius squared

- $\Omega = \frac{A}{r^2}$
- Sphere has 4π steradians



这里有个小插曲，因为我之前也不懂立体角所以贴进来了，哈哈。

这样看某个立体角的角度会对应多个不同的立体角。



$$\Phi = \int_{S^2} I \,\mathrm{d}\omega$$
$$= 4\pi I$$

$$I = \frac{\Phi}{4\pi}$$

最后一张图其实就是把之前的方程在均匀光源的特定解给解出来了。

给了个灯泡的具体例子，非常好。

做一下作业 5.

嗯在 Render 里的工作量好像太少了？反正用一下 Vector.hpp 里的函数就行了。

哈哈报警告了。

```
C:/Users/zhecai/Downloads/Assignment5/Code/Renderer.cpp: In member function 'void
Renderer::Render(const Scene&)':
C:/Users/zhecai/Downloads/Assignment5/Code/Renderer.cpp:215:11: warning: unused
variable 'scale' [-Wunused-variable]
  215 |     float scale = std::tan(deg2rad(scene.fov * 0.5f));
      |           ^~~~~
C:/Users/zhecai/Downloads/Assignment5/Code/Renderer.cpp:216:11: warning: unused
variable 'imageAspectRatio' [-Wunused-variable]
  216 |     float imageAspectRatio = scene.width / (float)scene.height;
      |           ^~~~~~~~~~~~~~~~
```

好吧发现注释写的很明白，直接告诉你怎么做了而且不用你去想原理。

毕竟 aspect ratio 好像既可以是宽高比，也可以是高宽比，这里按照注释可以推出是宽高比

```
// Also, don't forget to multiply both of them with the variable *scale*, and
// x (horizontal) variable with the *imageAspectRatio*
```

不是哥们，u 和 v 是啥呀?

然后我翻翻发现了 `st = st0 * (1 - uv.x - uv.y) + st1 * uv.x + st2 * uv.y;`

哦哦，原来是重心坐标参数啊，喵了个咪的，还得我自己找。

然后少安装了个 ubsan 库，嗯，这玩意似乎是 liunx 下的东西，所以嘛.....

```
9   target_link_libraries(RayTracing PUBLIC) #-fsanitize=undefined)
10
```

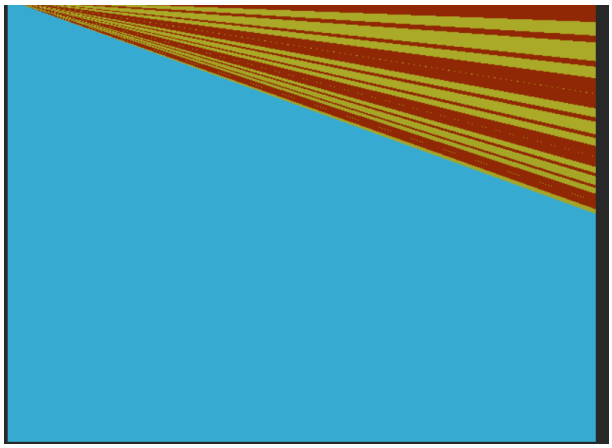
问题 输出 调试控制台 终端 端口 cmd

```
ult_deleteIS0_EEE7_M_headERKS4_]+0x7e): more undefined references to `__ubsan_handle_type_mismatch_v1' follow
C:/msys64/ucrt64/bin/../lib/gcc/x86_64-w64-mingw32/14.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe: CMakeFiles/RayTracing.dir/Renderer.c
pp.o:Renderer.cpp:(.text$_ZNSt27__uninitialized_default_n_1ILb0EE18__uninit_default_nIP8Vector3fyEET_S4_T0_[_ZNSt27__uninitialized_defaul
t_n_1ILb0EE18__uninit_default_nIP8Vector3fyEET_S4_T0_]+0x7e): undefined reference to `__ubsan_handle_pointer_overflow'
C:/msys64/ucrt64/bin/../lib/gcc/x86_64-w64-mingw32/14.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe: CMakeFiles/RayTracing.dir/Renderer.c
pp.o:Renderer.cpp:(.text$_ZNSt10_Head_baseILy0EP6ObjectLb0EE7_M_headERKS2_[_ZNSt10_Head_baseILy0EP6ObjectLb0EE7_M_headERKS2_]+0x30): unde
fined reference to `__ubsan_handle_type_mismatch_v1'
C:/msys64/ucrt64/bin/../lib/gcc/x86_64-w64-mingw32/14.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe: CMakeFiles/RayTracing.dir/Renderer.c
pp.o:Renderer.cpp:(.text$_ZNSt10_Head_baseILy0EP6ObjectLb0EE7_M_headERKS2_[_ZNSt10_Head_baseILy0EP6ObjectLb0EE7_M_headERKS2_]+0x56): unde
fined reference to `__ubsan_handle_type_mismatch_v1'
C:/msys64/ucrt64/bin/../lib/gcc/x86_64-w64-mingw32/14.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe: CMakeFiles/RayTracing.dir/Renderer.c
pp.o:Renderer.cpp:(.text$_ZNSt10_Head_baseILy0EP6ObjectLb0EE7_M_headERKS2_[_ZNSt10_Head_baseILy0EP6ObjectLb0EE7_M_headERKS2_]+0x7e): unde
fined reference to `__ubsan_handle_type_mismatch_v1'
C:/msys64/ucrt64/bin/../lib/gcc/x86_64-w64-mingw32/14.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe: CMakeFiles/RayTracing.dir/Renderer.c
pp.o:Renderer.cpp:(.text$_ZNSt11_Tuple_implILy0EJP5LightSt14default_deleteIS0_EEE7_M_headERKS4_[_ZNSt11_Tuple_implILy0EJP5LightSt14defaul
t_deleteIS0_EEE7_M_headERKS4_]+0x30): undefined reference to `__ubsan_handle_type_mismatch_v1'
C:/msys64/ucrt64/bin/../lib/gcc/x86_64-w64-mingw32/14.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe: CMakeFiles/RayTracing.dir/Renderer.c
pp.o:Renderer.cpp:(.text$_ZNSt11_Tuple_implILy0EJP5LightSt14default_deleteIS0_EEE7_M_headERKS4_[_ZNSt11_Tuple_implILy0EJP5LightSt14defaul
t_deleteIS0_EEE7_M_headERKS4_]+0x56): undefined reference to `__ubsan_handle_type_mismatch_v1'
C:/msys64/ucrt64/bin/../lib/gcc/x86_64-w64-mingw32/14.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe: CMakeFiles/RayTracing.dir/Renderer.c
pp.o:Renderer.cpp:(.text$_ZNSt11_Tuple_implILy0EJP5LightSt14default_deleteIS0_EEE7_M_headERKS4_[_ZNSt11_Tuple_implILy0EJP5LightSt14defaul
t_deleteIS0_EEE7_M_headERKS4_]+0x7e): more undefined references to `__ubsan_handle_type_mismatch_v1' follow
collect2.exe: error: ld returned 1 exit status
make[2]: *** [CMakeFiles/RayTracing.dir/build.make:132: RayTracing.exe] 错误 1
make[1]: *** [CMakeFiles/Makefile2:87: CMakeFiles/RayTracing.dir/all] 错误 2
make: *** [Makefile:91: all] 错误 2
```

喵的去掉了还不行给我报一堆错.....好吧是没去干净，一共要去掉两处。

```
target_compile_options(RayTracing PUBLIC -Wall -Wextra -pedantic -Wshadow -Wreturn-type) #-fsanitize=undefined)
target_compile_features(RayTracing PUBLIC cxx_std_17)
target_link_libraries(RayTracing PUBLIC) #-fsanitize=undefined)
```

生成了一个 .ppm 文件，用 VSCode 的插件看一下。



额……继续查错吧。

好吧看了下网上的代码是忘了变换……。

把 x，y 改成这样

```
float xmid = scene.width/2.0f, ymid=scene.height/2.0f;
x = (x - xmid) / (float)scene.width * scale * imageAspectRatio;
y = (y - ymid) / (float)scene.height  * scale;
```

但效果还是不太对……好吧作业要求的效果本来就不太对。要想让图正过来吧（y-ymid）改成（ymid-y）就行了